

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace Gameplay
{
    /* this is the score class that is used for dynamically generating the TAB image that is displayed to
    * the user when playing a song. Initially we used a single image for the source but due to the size
    * limitations of the 'int' that is used to determine the length of the image and the fact notes were
    * not being drawn past 10240px onwards we have developed a list of image sources that are og length
    * 10240 and repeat the viewport width of notes (at the end) on each image after the first one so
    * when you change to the next image you can purely show that image and not have to have a little
    * bit of the 1st and a little of the second. 1 this reduces the computation needed over a timer tick
    * when the images are swapping out that potentially leads to lag but also creates for a smooth looking
    * transition being able to swap to the next source completly. And exaple of this is:
    * if the viewport holds 4 beats and an image holds 12 the first image would hold numbers 1-12 the 2nd
    * would hold 8-20. this means when the first image gets to beat 8 all of the numbers 8-12 are displayed
    * the next beat the image sources are swapped, the image is scrolled and the numbers 9-13 are displayed
    * from the next source*/

    public class TScore
    {
        //The image list and current image that is being built
        public List<RenderTargetBitmap> renderImages;
        public RenderTargetBitmap renderImage;

        //the canvas and tools used to draw
        private Canvas toDrawTo;
        public DrawingVisual myDrawingVisual;
        public DrawingContext myDrawingContext;

        private Brush myBrush;
        private Point myLoc;
        private Pen myPen;

        //Constants used for the number of beats in a bar, the width and spacing of a note
        //the bar height, the starting position for where to draw the bar lines/
        const int NOTEWIDTH = 40;
        const int NOTESPACING = 24; //this allows for 12px either side of every note
        const int NBEATS = 4; //THIS WILL need to change depending on the time signiture so maybe its not a const.
        const int STARTXPOS = 0;
        const int STARTYPOS = 50;
        const int SCOREHEIGHT = 350;
        const int BAROVERLAP = 5;

        //This is the stock bar amount which is used to determine how many bars fit into
        //an image source (quarter notes = 40, 8th notes = 20, 16th = 10 ...)
        const int STOCKBARCOUNT = 40;
        public int numBarsPerImage;

        //bar size properties
        public int barSize;
        public int totalScoreLen;
        public int nBars;

        //listbox used for debugging purposes
        public ListBox printOutLoc;

        //current beat count information that is used for determining when to make a new image source
        public int currBeatCount;
        public int nextImageBeatCount;
        public int noteIndexForNextImage;
```

```

//index values used for the source images when built
public List<int> ScoreImageFlags;
public int flagIndex;
public int timeCountForRestartPos; //used for the time position of the note where the next draw will start from

//current song note index information
public int currNoteIndex;
public int totalSongNotes;

//flag used to signal the end of song
public bool endOfSong;

//Constructor setting all default values
public TScore(TSong theSong, Canvas toDrawTo /*, ListBox printOutLoc*/)
{
    numBarsPerImage = 0;
    noteIndexForNextImage = 0;
    totalSongNotes = 0;
    endOfSong = false;

    myBrush = new SolidColorBrush(Colors.White);
    myPen = new Pen(myBrush, 2);

    myDrawingVisual = new DrawingVisual();
    myDrawingContext = null;

    myLoc = new Point(); //Point used for when drawing the bar lines

    computeScoreSizes(theSong); //calculate the number of bar sizes
    createSources(theSong); //create the images

    //this.toDrawTo = toDrawTo; //draw the image out to the screen- debuggin purposes?
    //drawToScreen(toDrawTo);
}

//*****

/*Wrapper function to create the images sources*/
public void createSources(TSong theSong)
{
    //set the flag properties (when to change to the next image for viewport scrolling purposes
    //and where to continue drawing from on the next image create purposes
    ScoreImageFlags = new List<int>();
    flagIndex = 0;

    //beatcount start value;
    currBeatCount = 1;
    nextImageBeatCount = currBeatCount + 160; //160 because the viewport width is 11024 and 16 notes of 64 px fit in it

    ScoreImageFlags.Add(nextImageBeatCount - 16); //figure the first spot where the image will repeat from(1 viewport short)
    flagIndex++; //set the flag index (index into the array of locations) to the next for the next one to be added without overwriting the first

    currNoteIndex = 0; //starting index when looking into the song notes

    this.renderImages = new List<RenderTargetBitmap>(); //create the list of source images

    fillCurrentSourceImages(theSong); //function call to populate the needed smaller images into the source iamges array/list
}

//*****

/*Function used to create the smaller source images- length is 10240, 1 viewport width short of this
* they take the note index and current time values so that when the end of the image is reached
* on the next image they can start from here achieving the overlap to allow the sources to be straight
* swapped when the end if reached without loosing a few bars (1 viewport widths worth)*/
private void fillCurrentSourceImages(TSong theSong)
{
    //keep making images untill the end of song flag is raised
    while (!endOfSong)
    {
        //create a source image
    }
}

```

```

        renderImage = new RenderTargetBitmap(10240, SCOREHEIGHT, 96, 96, PixelFormats.Default);

        //draw background- testing purposes - the red
        drawBackground2();

        //draw the bar lines onto the iamge
        drawBarLines2();

        //draw notes onto render image
        drawNotes2(theSong);

        //add the currently drawn image to the sources list
        renderImages.Add(renderImage);
    }
}

//*****

/*This is the function that traverses the song list drawng the notes onto the image
 * source that is to be used for displaying the TAB*/
public void drawNotes2(TSong theSong)
{
    int offset = 12;        //half od the total note spacing/offset 12 either side

    //These are the variables which will store the string and fret information
    int theString = 0;
    String theFret = "-";
    String theFrets = ""; //used for output debugging

    int currentXLoc = STARTXPOS;        //this is the BASE location where the note will be displayed.
    int quarterNoteLen = (NOTEWIDTH + NOTESPACING);    //needs to be made a const.

    currentXLoc += offset;                //update the current XPos (start location in this instance)

    int duration = 0;                    //default a duration value

    bool EndOfNotesForImage = false;    //set the flag to signify the end of the legnth of this image to false
    int currTime = 0;

    //Loop while the flag to state the end of the image is not set to true
    while(!EndOfNotesForImage)
    {
        theFrets = "";                    //this is used for debugging

        //populate the time and duration information from the current song note
        duration = Convert.ToInt16(theSong.songNotes[currNoteIndex].theDuration);
        currTime = theSong.songNotes[currNoteIndex].theTime;

        //for each TDetail in the song note, draw to the screen (each string and fret position, caters for both single and chords)
        foreach (TNoteDetail t in theSong.songNotes[currNoteIndex].myNoteDetails)
        {
            theString = Convert.ToInt16(t.theString);    //converted to int as there is a case by number in the draw
            theFret = t.theFret;

            theFrets += theFret + ", ";                //this is a debugging string

            drawToImageHolder(theString, theFret, currentXLoc);
        }

        currBeatCount += duration;                //once the note has been drawn update the beatCount/Timer location by adding the notes duration

        //testing to see if the current note is to be used as the starting note for the next images (for overlap purposes)
        if ((currTime + 16) == (nextImageBeatCount))
        {
            noteIndexForNextImage = currNoteIndex;    //if so record the index so for the next image to draw from here
            timeCountForRestartPos = currTime;        //beatCout is also recorded as used to determine when the next next image will be
        }

        //check if beatcount is at end of current image to stop drawing and start the next image
        if (currBeatCount >= nextImageBeatCount)
            EndOfNotesForImage = true;
    }
}

```

```

        //output test
        //printOutLoc.Items.Add("Current time: " + currTime.ToString() + ". Duration: " + duration.ToString() + ". the frets:" + theFrets + ". At location: " + currentXLoc.ToString());

        //increment the xPos for the next note to draw at
        currentXLoc += (duration * quarterNoteLen);

        //increment the note index in the song
        currNoteIndex++;

        //if the last or past the last note in the song then signal the end of the current source and end of the song
        if (currNoteIndex >= totalSongNotes)
        {
            EndOfNotesForImage = true;
            endOfSong = true;
        }
    }

    nextImageBeatCount = timeCountForRestartPos + 160; //calculate the beat count for the end of the next piece

    ScoreImageFlags.Add(nextImageBeatCount - 16); //calculate the beatcount for the overlap start of the next iamge
    flagIndex++; //increment the flagIndex loc so the next value can be added in

    currentXLoc = 0; //reset the XLoc as will be drawing to a new image
    currNoteIndex = noteIndexForNextImage; //set the overlap index for the next image
    currBeatCount = theSong.songNotes[currNoteIndex].theTime; //reset the current beat count
}

//*****

/*This is the function used to draw the bar lines onto the image*/
private void drawBarLines2()
{
    //set the starting points of the lines and end points
    Point p1 = new Point(STARTXPOS, 50);
    Point p2 = new Point(renderImage.Width, 50);

    myDrawingContext = myDrawingVisual.RenderOpen();
    myDrawingContext.DrawLine(myPen, p1, p2);

    //draw the horizontal bar lines
    int incrementer = 50; //this is the vertical spacing const
    for (int i = 0; i < 5; i++)
    {
        p1.Y += incrementer;
        p2.Y += incrementer;

        myDrawingContext.DrawLine(myPen, p1, p2);
    }

    //draw the vertical bar lines
    p1 = new Point(STARTXPOS, 50);
    p2 = new Point(STARTXPOS, 300);

    //draw all of the end of bar bar lines
    int barIncrementer = barSize; //this is the bar width const.
    for (int i = 0; i < numBarsPerImage; i++)
    {
        p1.X += barIncrementer;
        p2.X += barIncrementer;

        myDrawingContext.DrawLine(myPen, p1, p2);
    }

    //draw onto the image itself
    myDrawingContext.Close();
    renderImage.Render(myDrawingVisual);
}

//*****

/*this is the function used to draw a red background onto the image, this is helpfull
*for testing purposes as in outside testing applications the background can be white
*and the text used to draw the notes in Jam sessions is white*/

```

```

public void drawBackground2()
{
    Brush tempBrush;

    tempBrush = new SolidColorBrush(Colors.Red);

    Rect myRect = new Rect();
    myRect.Height = 350;
    myRect.Width = 10240;

    myDrawingContext = myDrawingVisual.RenderOpen();
    myDrawingContext.DrawRectangle(tempBrush, myPen, myRect);
    myDrawingContext.Close();

    renderImage.Render(myDrawingVisual);
}

//*****
/*This is the function used to set all of the bar and image sizes for the class*/
public void computeScoreSizes(TSong theSong)
{
    int smallestDivision = theSong.smallestDivision;    //get the smallest division from the song
    nBars = theSong.numberOfBars;                       //get the number of bars in the song

    barSize = (NOTEWIDTH + NOTESPACING) * (smallestDivision * NBEATS); //determine the width of a single bar

    numBarsPerImage = STOCKBARCOUNT / smallestDivision; //calculate the number of bars per image source

    totalSongNotes = theSong.numberOfNotes;             //get the total number of song notes

    totalScoreLen = (barSize * numBarsPerImage);        //get the overall length needed for the whole score image (does note count overlaps)

    //testing message box
    //MessageBox.Show("smallest div" + smallestDivision.ToString() + "totalScoreLen: " + totalScoreLen.ToString());
}

//*****

/*This is the function used to draw the current notes 'detail' to the image,
 * each note can have multiple details (stings frets etc) to represent chords*/
public void drawToImageHolder( int theString, String theFret, int currentXLoc)
{
    bool rest = false; //default the rest flag

    switch (theString) //case of the string (which y location to draw at) setting the 'point' value for the draw later
    {
        case 1:
            myLoc = new Point(currentXLoc, 25);
            break;

        case 2:
            myLoc = new Point(currentXLoc, 75);
            break;

        case 3:
            myLoc = new Point(currentXLoc, 125);
            break;

        case 4:
            myLoc = new Point(currentXLoc, 175);
            break;

        case 5:
            myLoc = new Point(currentXLoc, 225);
            break;

        case 6:
            myLoc = new Point(currentXLoc, 275);
            break;

        default:
            //do nothing
            rest = true;
            break;
    }
}

```

```

    }

    //If the current note is not a rest then draw
    if (!rest)
    {
        //draw a text onto an image:
        FormattedText text = new FormattedText(theFret, new System.Globalization.CultureInfo("en-us"),
            FlowDirection.LeftToRight,
            new Typeface(new FontFamily("Arial"), FontStyles.Normal, FontWeights.Normal, new FontStretch()), 40, new SolidColorBrush(Colors.White));

        DrawingVisual drawingVisual = new DrawingVisual();
        DrawingContext drawingContext = drawingVisual.RenderOpen();
        drawingContext.DrawText(text, myLoc);          //draw the text at the point location calculate above
        drawingContext.Close();

        renderImage.Render(drawingVisual);          //draw onto the image source
    }
}

//*****

/*Testing function used to draw to the first/current image source determined by the index value to the screen*/
public void drawToScreen(Canvas toDrawTo)
{
    Image img = new Image();
    img.Source = renderImages[0];
    toDrawTo.Children.Add(img);
}
//*****
}

```